CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE251: PROGRAMMING IN JAVA

JAVA Exception Handling

DEPSTAR

Devang Patel Institute of Advance Technology and Research

Five keywords

1. Try block
2. Catch block
3. Finally block
4. Throw
5. Throws

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Advantage of Exception Handling

Why we use exception handling?

to maintain the normal flow of the application

Example

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

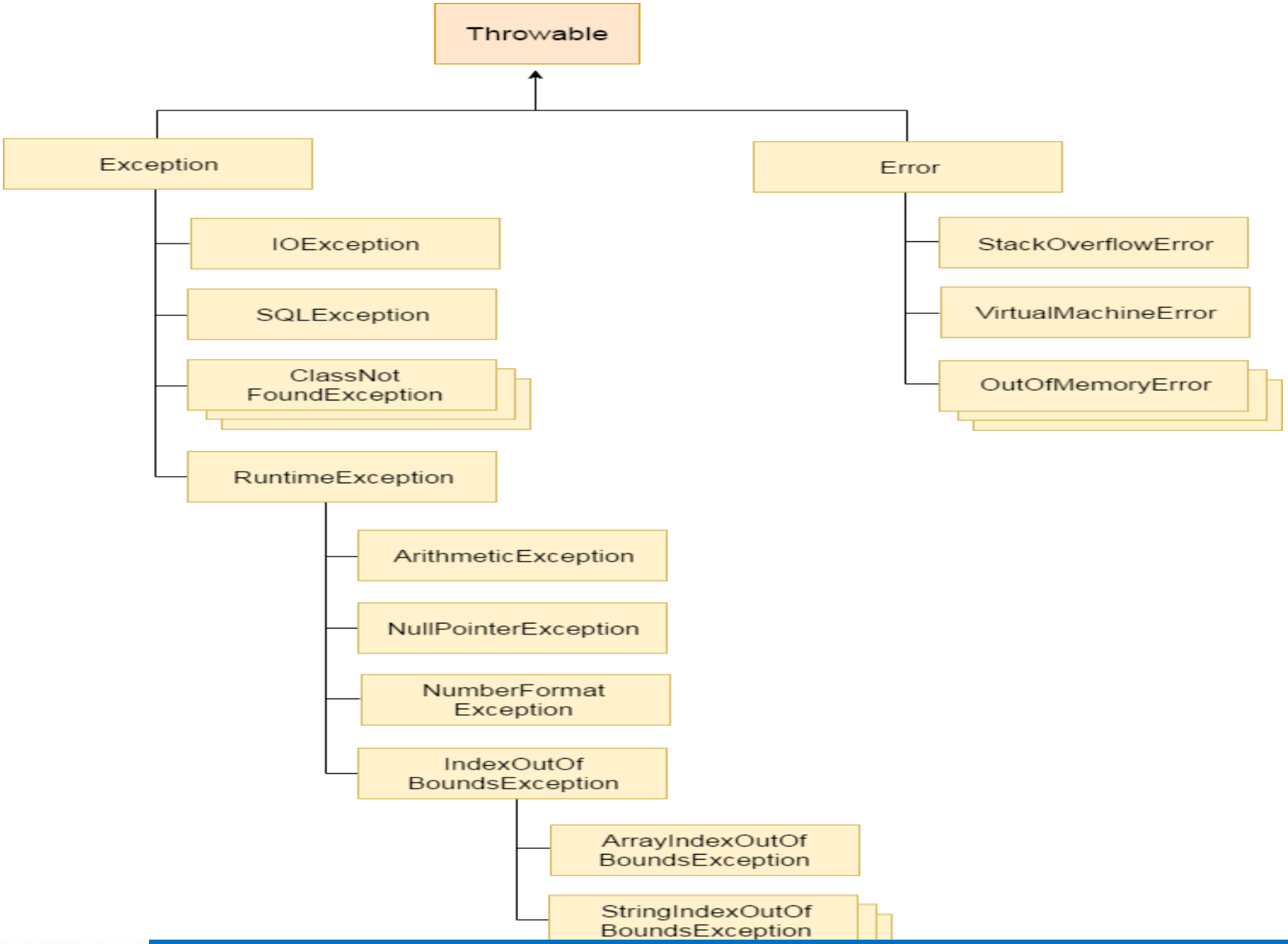
Meaning

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed.

If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Hierarchy of Java Exception classes



Types of Java Exceptions

1. Checked Exception
2. Unchecked Exception
3. Error

First Discuss Unchecked Exception

```
class UncheckedException
{
  P.S.V.M()
  {
        S.O.P("Hello");
        S.O.P("Hiiiii");
        S.O.P(10/0);            // arithmetic exception
        S.O.P("Hello");
        S.O.P("Hiiiii");
  }
}
```

Program will compile & executed abnormally

Unchecked Exception

The exception which are not checked by compiler is called unchecked exception.

Application contain unchecked exception code is compiled but run time JVM show exception.

We can handle such type of exception using try-catch or throws keyword

Example-2

```
class UncheckedException
{
   P.S.V.M()
   {
         int[] a = {10,20,30,40};
         S.O.P(a[0]);
         S.O.P(a[10]);                    // array index out of bound
   }                                      exception
}
```

This is the type of unchecked exception

Example-3

```
class UncheckedException
{
  P.S.V.M()
  {
        String str = “Charusat”;
        S.O.P(str.chaAt(2));
        S.O.P(str.chaAt(20);          // string index out of bound
                                      exception
  }
}
```

This is the type of unchecked exception

Example-4

```
class UncheckedException
{
  P.S.V.M()
  {
        String str = null;
        S.O.P(str.length());          // Null pointer exception
        }
}
```

This is the type of unchecked exception

checked exception

- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

- Java forces you to handle these error scenarios in some manner in your application code. They will come immediately into your face, once you start compiling your program.

Understand checked exception

```
public static void main(String[] args)
{
 FileReader file = new FileReader("somefile.txt");
}
```

//File not found exception

Program will not compile, compiler check exception and write try-catch block.

```java
public static void main(String[] args)
{
    try
    {
        FileReader file = new FileReader("somefile.txt");
    }
    catch (FileNotFoundException e)
    {
        //Alternate logic
        e.printStackTrace();
    }
}
```

More type of checked exception

- checked exceptions e.g. IOException, SQLException etc.
- Checked exceptions are checked at compile-time.

Error

- An error is considered as the unchecked exception.
- Error is irrecoverable
- e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Try-catch block

- Try block is used to enclose the code that might throw an exception
- Try block must be used within the method
- Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}
catch(Exception_class_Name ref){
}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}
finally{
}
```

Java catch block

- Java catch block is used to handle the Exception.
- It must be used after the try block only.
- You can use multiple catch block with a single try.

Example-1

```
public class TestTryCatch{
  public static void main(String args[]){
     int data=10/0;
     System.out.println("rest of the code...");
}
}
```

It will throw an exception-
Exception in thread main
java.lang.ArithmeticException:/ by zero

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac TestTryCatch.java

C:\Users\Administrator\Desktop\java>java TestTryCatch
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestTryCatch.main(TestTryCatch.java:3)

C:\Users\Administrator\Desktop\java>
```

Use try-catch block

```
public class TestTryCatch{
 public static void main(String args[]){
  try{
    int data=10/0;
  }
 catch(ArithmeticException  e)
  {       System.out.println(e);
  }
  System.out.println("rest of the code...");
}
}
```

By doing this we are able to execute the rest of the statements

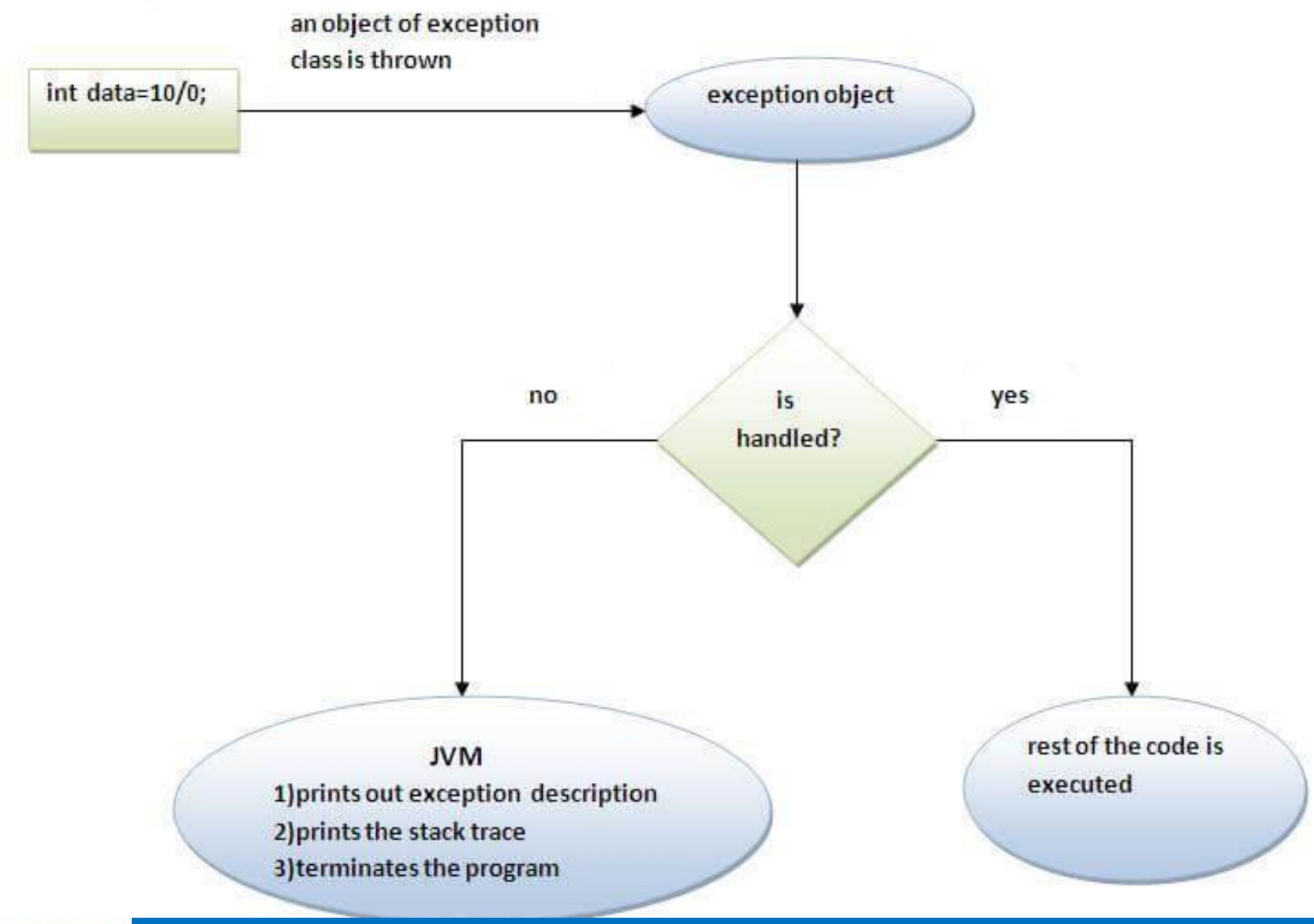
Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac try1.java

C:\Users\Administrator\Desktop\java>java try1
java.lang.ArithmeticException: / by zero
rest of the code...

C:\Users\Administrator\Desktop\java>_
```

Internal working of try-catch block



Java Multi catch block

- We want to handle different exception then use multi catch block

Rules

1. At a time only one exception is occurred & at a time only one catch block is executed
2. All catch block must be ordered from most specific to most general

Example-1

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;

  }
  catch(ArithmeticException e){
        System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){
       System.out.println("task 2 completed");}
  catch(Exception e){
      System.out.println("common task completed");}
   System.out.println("rest of the code...");
 }
}
```

Output

```
task1 completed
rest of the code...
```

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac try4.java

C:\Users\Administrator\Desktop\java>java try4
task1 is completed
rest of the code...

C:\Users\Administrator\Desktop\java>
```

Example-2 What is the output here

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
catch(Exception e){
      System.out.println("common task completed");}
  catch(ArithmeticException e){
       System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){
      System.out.println("task 2 completed");}
System.out.println("rest of the code...");
 }
}
```

Output

Compile-time error

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac try3.java
try3.java:9: error: exception ArithmeticException has already been caught
   catch(ArithmeticException e){
   ^
try3.java:11: error: exception ArrayIndexOutOfBoundsException has already been c
aught
   catch(ArrayIndexOutOfBoundsException e){
   ^
2 errors

C:\Users\Administrator\Desktop\java>_
```

```
public class MultipleCatchBlock2 {

   public static void main(String[] args) {

        try{
            int a[]=new int[5];

          System.out.println(a[10]);
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
         System.out.println("rest of the code");
   }
}
```

```
public class MultipleCatchBlock3 {

  public static void main(String[] args) {

      try{
         int a[]=new int[5];
         a[5]=30/0;
         System.out.println(a[10]);
        }
        catch(ArithmeticException e)
          {
           System.out.println("Arithmetic Exception occurs");
          }
        catch(ArrayIndexOutOfBoundsException e)
          {
           System.out.println("ArrayIndexOutOfBounds Exception occurs");
          }
        catch(Exception e)
          {
           System.out.println("Parent Exception occurs");
          }
        System.out.println("rest of the code");
  }
}
```

```
public class MultipleCatchBlock4 {

  public static void main(String[] args) {

      try{
         String s=null;
         System.out.println(s.length());
        }
        catch(ArithmeticException e)
          {
           System.out.println("Arithmetic Exception occurs");
          }
        catch(ArrayIndexOutOfBoundsException e)
          {
           System.out.println("ArrayIndexOutOfBounds Exception occurs");
          }
        catch(Exception e)
          {
           System.out.println("Parent Exception occurs");
          }
        System.out.println("rest of the code");
  }
}
```

How to handle multiple exception?

Using Nested try block- every try-catch block cause separate error

Java Nested try block

```
try
{
   statement 1;
   statement 2;
   try
   {
      statement 1;
      statement 2;
   }
   catch(Exception e)
   {
   }
}
catch(Exception e)
{
}
```

Example

```
class MultiExcep{
 public static void main(String args[]){
  try{
    try{
     System.out.println("divide by zero exception");
     int b =10/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}
    System.out.println("normal flow..");
 }
}
```

Java finally block

The finally block is always executed irrespective of try-catch block

It is used *to execute important code* such as closing connection, stream etc.- called cleanup process

Means-

1. To close the database connection
2. To destroy the objects
3. To close the file or channel

Case-1 (Normal termination)

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output: 5
finally block is always executed
rest of the code...

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f1.java

C:\Users\Administrator\Desktop\java>java f1
5
finally block is always executed
rest of the code...

C:\Users\Administrator\Desktop\java>_
```

Case-2 (normal termination)

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=10/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f2.java

C:\Users\Administrator\Desktop\java>java f2
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...

C:\Users\Administrator\Desktop\java>
```

Case-3 (abnormal termination)

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=10/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Divide by zero exception but catch not matched
Finally block will be execute

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f3.java

C:\Users\Administrator\Desktop\java>java f3
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at f3.main(f3.java:4)

C:\Users\Administrator\Desktop\java>_
```

Case-4 (abnormal termination)

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=10/0;
   System.out.println(data);
  }
  catch(ArithmaticException e){System.out.println(10/0);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Divide by zero exception in try and catch
Finally block will be execute

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f4.java

C:\Users\Administrator\Desktop\java>java f4
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at f4.main(f4.java:9)

C:\Users\Administrator\Desktop\java>_
```

Case-5 (abnormal termination)

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=10/0;
   System.out.println(data);
  }
  catch(ArithmaticException e){System.out.println(e);}
  finally{
  System.out.println(10/0);}
  System.out.println("rest of the code...");
  }
}
```

Divide by zero exception catch block execute Finally block will be executed by AE

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f6.java

C:\Users\Administrator\Desktop\java>java f6
java.lang.ArithmeticException: / by zero
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at f6.main(f6.java:9)

C:\Users\Administrator\Desktop\java>_
```

Case-6 (normal termination)

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=100/10;
   System.out.println(data);
  }

  finally{
  System.out.println("finally block");}
  System.out.println("rest of the code...");
  }
}
```

Try block
Finally block

Conclusion

In all the cases finally block will be executed

Find Two Cases?

Two cases finally block will not be executed

Case-1

System.exit(0);

exit()- JVM is shutdown so finally block will not be executed

Example-1

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
   System.exit(0);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output: 5

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f7.java

C:\Users\Administrator\Desktop\java>java f7
5

C:\Users\Administrator\Desktop\java>_
```

Example-2

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
   System.exit(0);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac f8.java

C:\Users\Administrator\Desktop\java>java f8
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...

C:\Users\Administrator\Desktop\java>_
```

Case-2

Output:
Exception in thread "main"
java.lang.ArithmeticException: / by zero

```
class TestFinallyBlock{
  public static void main(String args[]){
  System.out.println(25/0);
  try{
     System.out.println(“try block”);
}
  catch(ArithmeticException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Will it Compile?

Output:
error: 'try' without 'catch', 'finally' or resource declarations

```
class TestFinallyBlock{
  public static void main(String args[]){
try{
     System.out.println(“try block”);
}
 System.out.println(25/0);
  catch(ArithmeticException e){System.out.println(e);}
  finally{
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Exception Methods

To print the exception information, we have 3 methods

1. toString()
2. getMessage()
3. printStackTrace()

toString()

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
}
  catch(ArithmeticException e){
   System.out.println(e.toString());
}
  finally{
  System.out.println("finally block is always executed");}
}
}
```

Output:
java.lang.ArithmeticException: / by zero
finally block is always executed

getMessage()

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
}
  catch(ArithmeticException e){
   System.out.println(e.getMessage());
}
  finally{
  System.out.println("finally block is always executed");}
}
}
```

Output:
/ by zero
finally block is always executed

printStackTrace()

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
}
  catch(ArithmeticException e){
   e.printStackTrace();
}
  finally{
  System.out.println("finally block is always executed");}
}
}
```

Output:
finally block is always executed
java.lang.ArithmeticException: / by zero

printStackTrace()

- **printStackTrace**() helps the programmer to understand where the actual problem occurred. It helps to trace the exception.
- It **is printStackTrace**() method of Throwable class inherited by every exception class.
- This method prints the same message of e object and also the line number where the exception occurred.

Second approach of exception handling

throws

throws

Throws keyword is delegating exception responsibility to caller method.

Syntax

return_type method_name() **throws** exception_class_name{

//method code

}

Which type of exception should be declared?

Only checked exception

Why?

Checked exception are not in our control

Advantage of Java throws keyword

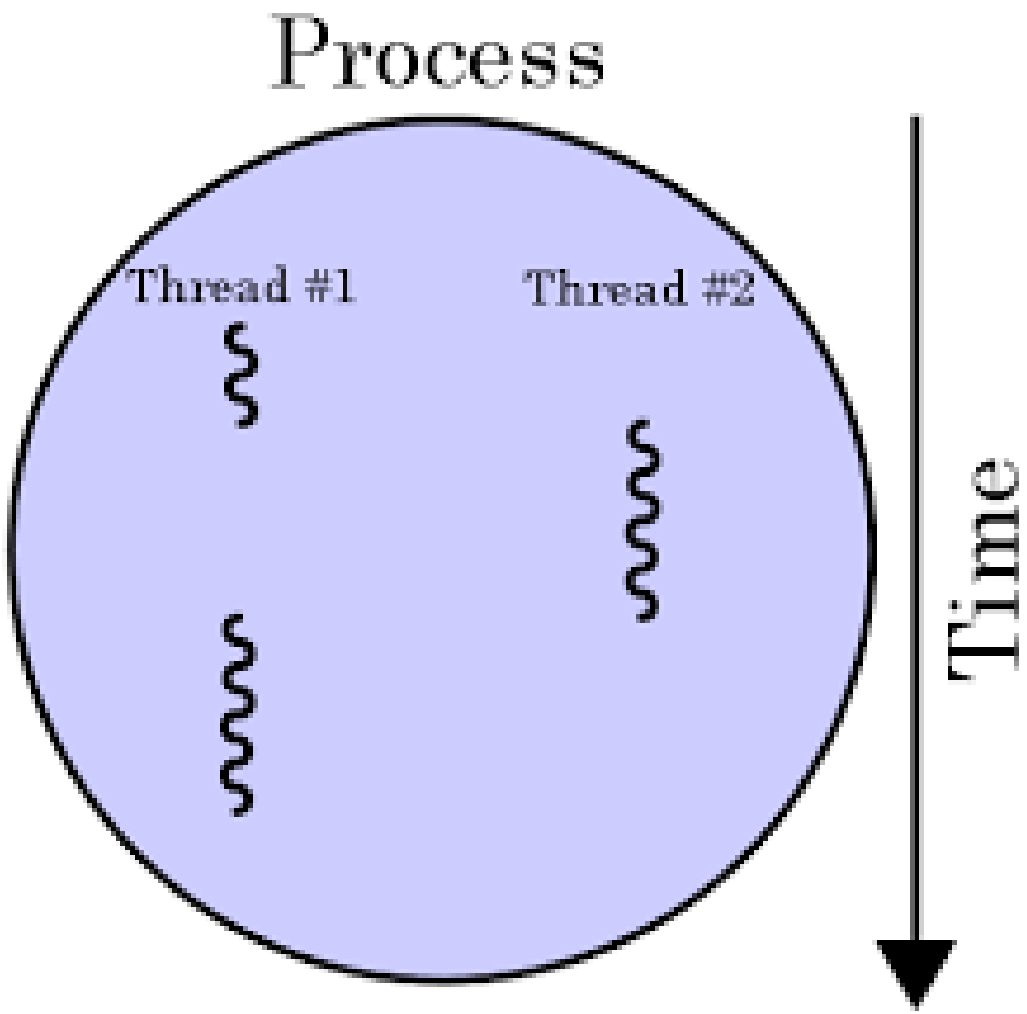
- Now Checked Exception can be propagated (forwarded in call stack).
- It provides information to the caller of the method about the exception.

Example

```
public class MyClass {
  void m1()
  {
     m2();
  }
  void m2()
  {
     m3();
  }
P.S.V.M() {
    new MyClass().m1();
   }
}
```

```
void m3()
 {
    S.O.P("sleeping mode");
    Thread.sleep(3000);
    S.O.P("awake mode");
  }
```

Output-
error: unreported exception InterruptedException; must be caught or declared to be thrown
Thread.sleep(3000);


Process
Thread #1
Thread #2
Time

Handle it by try-catch

```
public class MyClass {
  void m1()
  {
     m2();
  }
  void m2()
  {
     m3();
  }
P.S.V.M() {
    new MyClass().m1();
   }
}
```

```
void m3()
 {
    try{
    S.O.P("sleeping mode");
    Thread.sleep(3000);
    S.O.P("awake mode");
    }
    catch(InterruptException i)
   {
     i.printStackTrace();
    }
 }
```

Output-
sleeping mode
awake mode

How to use throws keyword here?

If m3() method is not able to handle the Interrupted exception then it will throws to the m2() method.

Bcs m2() is caller method

Handle it by throws

```
public class MyClass {
 void m1()
 {
    m2();
 }
 void m2()
 {
    try {   m3(); }
   catch(InterruptException i)
   {
     i.printStackTrace();
    }
 }
P.S.V.M() {
   new MyClass().m1();
  }
}
```

```
 void m3() throws
InterruptedException
 {
    S.O.P("sleeping mode");
    Thread.sleep(3000);
    S.O.P("awake mode");

}
```

Output-
sleeping mode
awake mode

If m2() method is not able to handle the Interrupted exception then it will throws to the m1() method.

Propagate throws

```
public class MyClass {
 void m1()
 {
    try{  m2(); }
   catch(InterruptException i)
   {
     i.printStackTrace();
    }
 }
 void m2() throws
InterruptedException
 {
    m3();
}
P.S.V.M() {
   new MyClass().m1();
  }
}
```

```
 void m3() throws
InterruptedException
 {
    S.O.P("sleeping mode");
    Thread.sleep(3000);
    S.O.P("awake mode");

}
```

Output-
sleeping mode
awake mode

If m1() method is not able to handle the Interrupted exception then it will throws to the main()method.

Propagate throws

```
public class MyClass {
  void m1() throws
InterruptedException
  {
      m2();
}
  void m2() throws
InterruptedException
  {
     m3();
}
P.S.V.M() {
try{
      new MyClass().m1();
}
catch(InterruptException i)
    {
       i.printStackTrace();
     }
   }
}
```

```
 void m3() throws
InterruptedException
  {
     S.O.P("sleeping mode");
     Thread.sleep(3000);
     S.O.P("awake mode");

}
```

Output-
sleeping mode
awake mode

If main() method is not able to handle the Interrupted exception then it will throws to the JVM.

Propagate throws

```
public class MyClass {
  void m1() throws
InterruptedException
  {
      m2();
}
  void m2() throws
InterruptedException
  {
     m3();
}
P.S.V.M() throws
InterruptedException
{
    new MyClass().m1();
}
}
```

```
 void m3() throws
InterruptedException
  {
     S.O.P("sleeping mode");
     Thread.sleep(3000);
     S.O.P("awake mode");

}
```

Output-
sleeping mode
awake mode

Now discuss last keyword

throw

Purpose of throw keyword

Understand it with example-

Arithmetic Exception-

1. predefine exception
2. JVM is creating object of Arithmetic Exception
3. JVM is printing exception information:

/ by zero

What about user define ArithmeticException?

User define exception

User define information

How to create user object with information?

```
new ArithmeticException("CHARUSAT / by zero");
```

How to handover user object to JVM?

By using throw keyword

throw new ArithmeticException("CHARUSAT / by zero");

Example-1

```
public class MyClass {

    public static void main(String args[]) {

        System.out.println("user  define message");

        throw new ArithmeticException("CHARUSAT / by zero");

    }
}
```

Output-
user define message
Exception in thread "main" java.lang.ArithmeticException: CHARUSAT / by zero

Example-2

```
public class MyClass {

   public static void main(String args[]) {

      System.out.println("user define message");

      try{
            throw new Exception();
         }
      catch(Exception e){
            System.out.println(e.getMessage());
         }
   }
}
```

Output-
user define message
null

Example-3

```
public class MyClass {

   public static void main(String args[]) {

      System.out.println("user define message");

      try{
            throw new Exception("base exception class message");
         }
      catch(Exception e){
            System.out.println(e.getMessage());
         }
   }
}
```

Output-
user define message
base exception class message

Example-4

```
public class MyClass {

   public static void main(String args[]) {

      System.out.println("user define message");

     throw new ArithmeticException("CHARUSAT / by zero");

     System.out.println("rest of the code...");

    }
}
```

Output-
error: unreachable statement
System.out.println("user define message");

Reason

The lines after a ‘throw’ will never get executed.

We only use throw when we want to specify a restriction to the application we're working on.

anything after throw statement will not execute

How to create user define exception?

Two Types

1. User define checked exception
   a) default constructor approach
   b) parameterized constructor approach
2. User define unchecked exception
   a) default constructor approach
   b) parameterized constructor approach

1.
create user defined checked exception by using default constructor approach

1. create the exception class first
2. use the exception class in project

Example-1

```
class MyException extends Exception{
 // default constructor generated by compiler
}

public class MyClass {
   public static void main(String args[]) {
     try{
            throw new MyException();
         }
      catch(MyException e){
            System.out.println(e.getMessage());
           System.out.println("exception caught successfully");
         }
   }}
```

Output-
null
exception caught successfully

2.
create user defined checked exception by using parameterized constructor approach

Example-1

```
class MyException extends Exception{
        MyException(String s)    {        super(s);        }
}

public class MyClass {

  public static void main(String args[]) {
    try{
        throw new MyException("Java user defined exception");
      }
    catch(MyException e){
        System.out.println(e.getMessage());
       System.out.println("exception caught successfully");
      }
}}
```

Output-
Java user defined exception
exception caught successfully

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLExcept ion.

Homework-1

Design an InvalidAgeException class and throw the exception if age is less then 18.

The exception will print the information

"not eligible for voting in 2019"

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
```

```
class custom{

  static void validate(int age)throws
InvalidAgeException{
    if(age<18)
     throw new InvalidAgeException("not valid")
    else
     System.out.println("welcome to vote");
  }

  public static void main(String args[]){
     try{
     validate(13);
     }catch(Exception
m){System.out.println("Exception occured:
"+m);}

     System.out.println("rest of the code...");
  }
}
```

Administrator: Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd desktop

C:\Users\Administrator\Desktop>cd java

C:\Users\Administrator\Desktop\java>javac custom.java

C:\Users\Administrator\Desktop\java>java custom
Exception occured: InvalidAgeException: not valid
rest of the code...

C:\Users\Administrator\Desktop\java>
```

Homework-2

create user defined unchecked exception by using default constructor & parameterized approach

```
class UserDefinedException extends
RuntimeException {
  UserDefinedException(String s) {
      super(s);
  }
}
```

Any Question??

Q-01

What is the output of this program?

```
1.class exception_handling
2.{
3.public static void main(String args[])
4.{
5.try
6.{
7.System.out.print("Hello" + " " + 1 / 0);
8.}
9.catch(ArithmeticException e)
10.{
11.System.out.print("World");
12.}
13.}
14.}
```

Q-02

Given:

```
try { int x = Integer.parseInt("two"); }
```

Which could be used to create an appropriate catch block? (Choose all that apply.)

Options are-

A. ClassCastException

B. IllegalStateException

C. NumberFormatException

D. IllegalArgumentException

E. ExceptionInInitializerError

F. ArrayIndexOutOfBoundsException

3

Which of these keywords is used to manually throw an exception?

a) try
b) finally
c) throw
d) catch

4

What is the output of this program?

```
1.class exception_handling
2.{
3.public static void main(String args[])
4.{
5.try
6.{
7.int a, b;
8.b = 0;
9.a = 5 / b;
10.System.out.print("A");
11.}
12.catch(ArithmeticException e)
13.{
14.System.out.print("B");
15.}
16.}
17.}
```

5. What is the output here

```
1.class exception_handling
2.{
3.public static void main(String args[])
4.{
5.try
6.{
7.int a, b;
8.b = 0;
9.a = 5 / b;
10.System.out.print("A");
11.}
12.catch(ArithmeticException e)
13.{
14.System.out.print("B");
15.}
16.finally
17.{
18.System.out.print("C");
19.}
20.}
```

Q-06

```
try
{
   int x = 0;
   int y = 5 / x;
}
catch (Exception e)
{
   System.out.println("Exception");
}
catch (ArithmeticException ae)
{
   System.out.println(" Arithmetic Exception");
}
System.out.println("finished");
```

A.Finished
B. Exception
C. Compilation fails.
D. Arithmetic Exception

```
public class X
{
   public static void main(String [] args)
   {
      try
      {
         badMethod();
          System.out.print("A");
      }
     catch (Exception ex)
      {
          System.out.print("B");
      }
      finally
      {
          System.out.print("C");
      }
      System.out.print("D");
  }
   public static void badMethod()
   {
      throw new Error(); /* Line 22 */
   }
}
```

- ABCD
- Compilation fails.
- C is printed before exiting with an error message.
- BC is printed before exiting with an error message.

```
public class X {
public static void main(String [] args)
{ try
{ badMethod();
System.out.print("A");
 }
catch (RuntimeException  ex) /* Line 10 */
{ System.out.print("B");
}
catch (Exception ex1)
{
System.out.print("C");
 }
finally {
System.out.print("D");
 }
System.out.print("E");
}
public static void badMethod()
 {
throw new RuntimeException();
} }
```

- BD
- BCD
- BDE
- BCDE

- https://www.aliensbrain.com/quiz_events/0

Wrapper class

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
- Following are the features used for this:
- **Autoboxing**
- **unboxing**
- The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

- Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.

- **Synchronization:** Java synchronization works with objects in Multithreading.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitl
y
Integer j=a;//autoboxing, now compiler will write Integer.valueOf
(a) internally
 System.out.println(a+" "+i+" "+j);
}}
```

Output:20 20 20

Administrator: Command Prompt

```
C:\Users\Administrator\Desktop\java>javac wc1.java

C:\Users\Administrator\Desktop\java>java wc1
20 20 20

C:\Users\Administrator\Desktop\java>_
```

Unboxing

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

 System.out.println(a+" "+i+" "+j);
}}
```

Administrator: Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd desktop

C:\Users\Administrator\Desktop>cd java

C:\Users\Administrator\Desktop\java>javac wc2.java

C:\Users\Administrator\Desktop\java>java wc2
3 3 3

C:\Users\Administrator\Desktop\java>
```